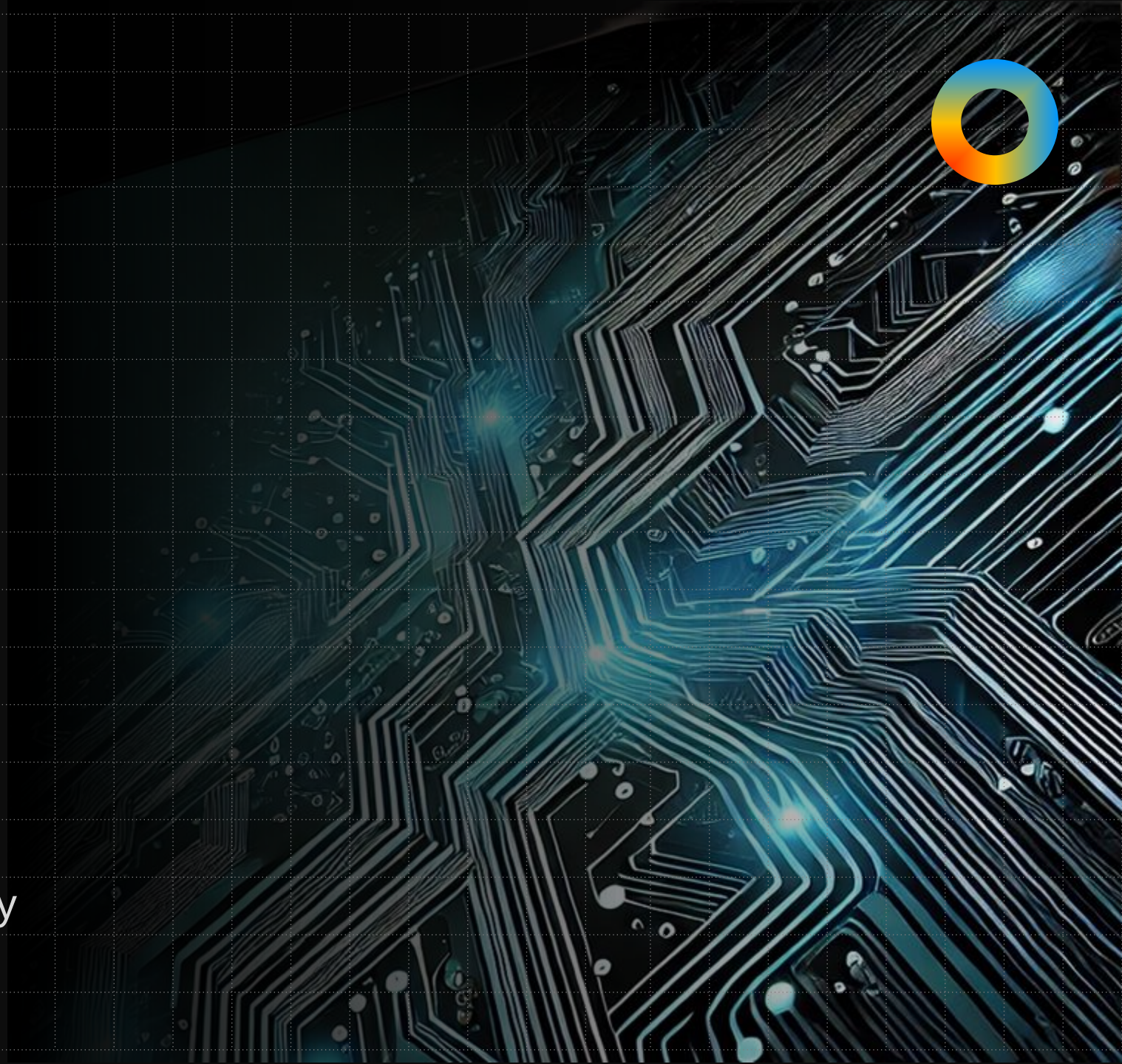
Python Programming

Numpy

Dr. Chun-Hsiang Chan
Department of Geography
National Taiwan Normal University



Outlines

- Indexing
- Dtype
- Dimension
- Shape
- Reshape
- Concatenation
- Split
- Sort
- Random
- Statistics
- Normal Distribution
- Central Limit Theory

NumPy

- NumPy is a powerful package for data processing because one of the most important function in NumPy – ndarray is developed along with the nature of CPU architecture, which leverages continuous memory for storing array (unlike list).
- Why NumPy is always faster than Python list? Because only a few is written in Python and most of parts (need powerful computation and fast calculation) are written in C or C++.
- Before we start, here, let's tell you how to call a Python library.

```
# import python packages
import numpy as np # simplify numpy into np
a = np.ones((3,2))
```

```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

NumPy – Indexing

- The most powerful part in NumPy is array and its calculation; therefore, we will introduce several functions for NumPy's array. First of all, we introduce the simpliest way to declare a NumPy array.

```
# declare a numpy array
arr = np.array([0, 1, 2, 3, 4])
print(arr)
print(arr[3])
print(type(arr))
mylist = [0, 1, 2, 3, 4]
arr_ = np.array(mylist) # Is there any difference between arr and arr_?
```

```
[0 1 2 3 4]
3
<class 'numpy.ndarray'>
```

```
array([0, 1, 2, 3, 4])
```

NumPy – Indexing

- Other approaches to declare a NumPy array.

```
# declare a numpy array
arr1 = np.array([4])        array([4])
arr2 = np.array(4)          array(4)
arr3 = np.array((4))        array(4)
arr4 = np.array((4,4))      array([4, 4])
# what is the difference among arr1, arr2, arr3, and arr4?
# tell me your finding…
```

NumPy – Indexing

- After we introduce the 0-dimensional and 1-dimensional array, it is time for 2-dimensional and 3-dimensional or even higher dimensional arrys.

```
# declare a numpy array
arr2D = np.array([[1, 2, 3],[4, 5, 6]])
arr3D = np.array([[[1, 2, 3],[4, 5, 6]], [[7, 8, 9],[10, 11, 12]]])
# indexing
arr2D[1, 2] = ?
arr3D[1, 0, 2] = ?
arr3D[-1, -2, -3] = ?
# what if we want to obtain 11 in the arr3D, then how can we index it?
```

NumPy – Indexing

- Advance indexing in NumPy.

```
# indexing
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
# try these codes
print(arr[:3])                  [1 2 3]
print(arr[::2])                 [1 3 5 7 9]
print(arr[0:8:2])               [1 3 5 7]
print(arr[::-1])                [10  9  8  7  6  5  4  3  2  1]
# 2-D array
arr_ = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr_[1:2, 0:2])           [[4 5]]
```

NumPy – Indexing

- Filter is very important in data cleaning and data preprocessing.

```
# design a filter in the array
arr = np.array([11, 24, 35, 24, 55, 62, 71, 84, 29, 10, 61, 42])
print(arr>50)
[False False False False  True  True  True  True False False  True False]
print(arr[arr>50])
[55 62 71 84 61]
# what is the difference between arr>50 and arr[arr>50]?
# tell me your idea…
```

NumPy – Indexing

- Sometimes, you want to the special value in your array.

```
# search in the array
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
x = np.where(arr == 3)
print(x) # I perfer to use x[0], and do you why?

# advance search
y = np.where(arr%2 == 0)
print(y)
```

NumPy – Dtype

- NumPy array supports several data types.

- `i` integer
- `b` Boolean
- `u` unsigned integer
- `f` float
- `c` complex float
- `m` timedelta
- `M` datetime
- `O` object
- `S` string
- `U` unicode string

```
# example 1
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print(arr.dtype)    int64
# example 2
arrs = np.array(['apple','ball','cyan'])
print(arrs.dtype)   <U5
# example 3
arr1 = np.array([1, 2, 3, 4, 5, 6], dtype='f')
print(arr1.dtype)   float32
# try this  array([b'1.0', b'2.0', b'3.0', b'4.0', b'5.0', b'6.0'], dtype='|S32')
arr1.astype('S')
```

NumPy – Dimension

- Confirming the dimension of an array is crucial, especially in deep learning model because we leverage several arrays for calculation; however, their dimensions usually are different.

```
# confirm the array dimension
print(arr2D.ndim)  2
print(arr3D.ndim)  3
arr5 = np.array([1, 2, 3, 4], ndmin=5)
# print array
print(arr5)                                    [[[[[1 2 3 4]]]]]
print('number of dimensions :', arr5.ndim) number of dimensions : 5
```

NumPy – Shape

- When you need to duplicate an array, you need to remember that using copy function instead of directly assigning.
- Another issue is how get the dimension information in an array. Here, we introduce a new function – **shape**.

```
# copy an array
arr = np.array([1, 2, 3, 4])
b = arr.copy()
arr[2] = 100
# observe the array
print(arr)
print(b)
```

```
# print the shape of an array
print(arr.shape())      (4,)
# print the specfic dimension
arr2 = np.array([[1,2],[3,4],[5,6]])
print(arr2.shape)       (3, 2)
print(arr2.shape[1])    2
```

NumPy – Reshape

- Sometimes, we want to modify the shape of an array.

```
# reshape
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
# 1-D to 2-D
arr_ = arr.reshape(2, 6)
print(arr_)
# 1-D to 3-D
arr_1 = arr.reshape(2, 3, 2)
print(arr_1)
# try this
arr_2 = arr_1.reshape(-1)
print(arr_2)
```

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

NumPy – Concatenation

- Here, we introduce how to join one array to another array.

```
# concat two arrays
arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
arr2 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
arr_0 = np.concatenate((arr1, arr2))
print(arr_0)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12]
```

```
# concat two arrays
arr3 = np.array([[1, 2, 3, 4, 5, 6], [ 7, 8, 9, 10, 11, 12]])
arr4 = np.array([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]])
arr_1 = np.concatenate((arr3, arr4), axis=0)
arr_2 = np.concatenate((arr3, arr4), axis=1)
print(arr_1)
print(arr_2)
```

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
[[ 1  2  3  4  5  6  1  2  3  4  5  6]
 [ 7  8  9 10 11 12  7  8  9 10 11 12]]
```

NumPy – Split

- You may join one array to another, so you may split one array into several sub-arrays.

```
# split into three arrays
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
arr1 = np.array_split(arr, 3)
print(arr1)
arr2 = np.array([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]])
arr3 = np.array_split(arr2, 3, axis=1)
arr4 = np.array_split(arr2, 2, axis=0)
print(arr3)
print(arr4)
```

```
[array([1, 2, 3, 4]), array([5, 6, 7, 8]), array([ 9, 10, 11, 12])]
```

```
[array([[1, 2],
        [7, 8]]), array([[ 3,  4],
        [ 9, 10]]), array([[ 5,  6],
        [11, 12]])]
[array([[1, 2, 3, 4, 5, 6]]), array([[ 7,  8,  9, 10, 11, 12]])]
```

NumPy – Sort

- Sorting is one of the most important function in the numerical analysis.

```
# sort 1-D array
arr1 = np.array(['x','t','a','h','z'])
arr2 = np.array([True, False, False, True])
print(np.sort(arr1))
print(np.sort(arr2))

# sort 2-D array
arr3 = np.array([[33, 5, 4],[42, 105, 78]])
print(np.sort(arr3))
```

```
['a' 'h' 't' 'x' 'z']
[False False  True  True]
[[  4   5  33]
 [ 42  78 105]]
```

NumPy – Random

- Random variable generation

```
from numpy import random
x1 = random.randint(100)
print(x1)
x2 = random.randint(100, size=(2, 4))
print(x2)
y = random.rand(5, 2)
print(y)
z = random.choice([1,2,3,4], p=[0.5, 0.1, 0.2, 0.2], size=(2,5))
print(z)
# observe the differences among these codes
```

```
41
[[56 17 47 94]
 [82 69 64 65]]
[[0.25828183 0.85311945]
 [0.36852578 0.84986729]
 [0.28646403 0.82969085]
 [0.50318924 0.52606354]
 [0.49049719 0.1397218 ]]
[[1 2 1 3 1]
 [1 2 1 1 1]]
```

NumPy – Random

- Imagine that you are playing cards, so you need to …

```
arr = np.array([1, 2, 3, 4, 5])
# using shuffle
random.shuffle(arr)
print(arr) ──────────────────────────────→ [3 5 4 1 2]
# using permutation
print(random.permutation(arr))   [2 3 1 5 4]
# observe the differences among these codes
```

NumPy – Statistics

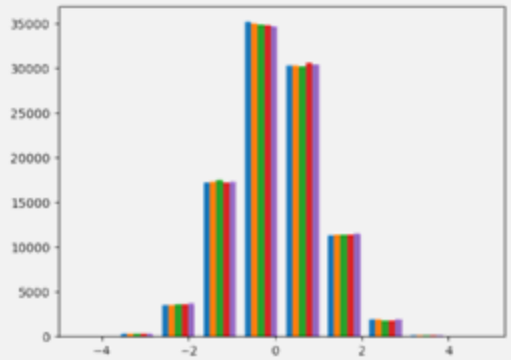
- Simple descriptive statistical analysis for an array.

```
arr = np.array([11, 24, 35, 24, 55, 62, 71, 84, 29, -10, -61, 42])
# get simple descriptive statistics
print(np.mean(arr)) # mean
print(np.var(arr)) # variance
print(np.std(arr, ddof=1)) # standard deviation
print(np.median(arr)) # median
print(np.absolute(arr)) # absolute
# create a range
print(np.arange(10))
print(np.arange(0, 100, 10)) # from 0 to 100 with 10-step hopping
```

NumPy – Normal Distribution

- In the statistics, we have to use random variable from specific distribution, such as, normal, binomial, and uniform distribution.

```
# normal distribution
nor1 = random.normal(size=(100000, 3))
print(nor1)
# given an average and standard deviation
nor2 = random.normal(loc=5, scale=1, size=(100000, 3))
print(nor2)
# but how to show whether they are normal distribution?
import matplotlib.pyplot as plt
plt.hist(nor1) # you may try nor2
plt.show()
```



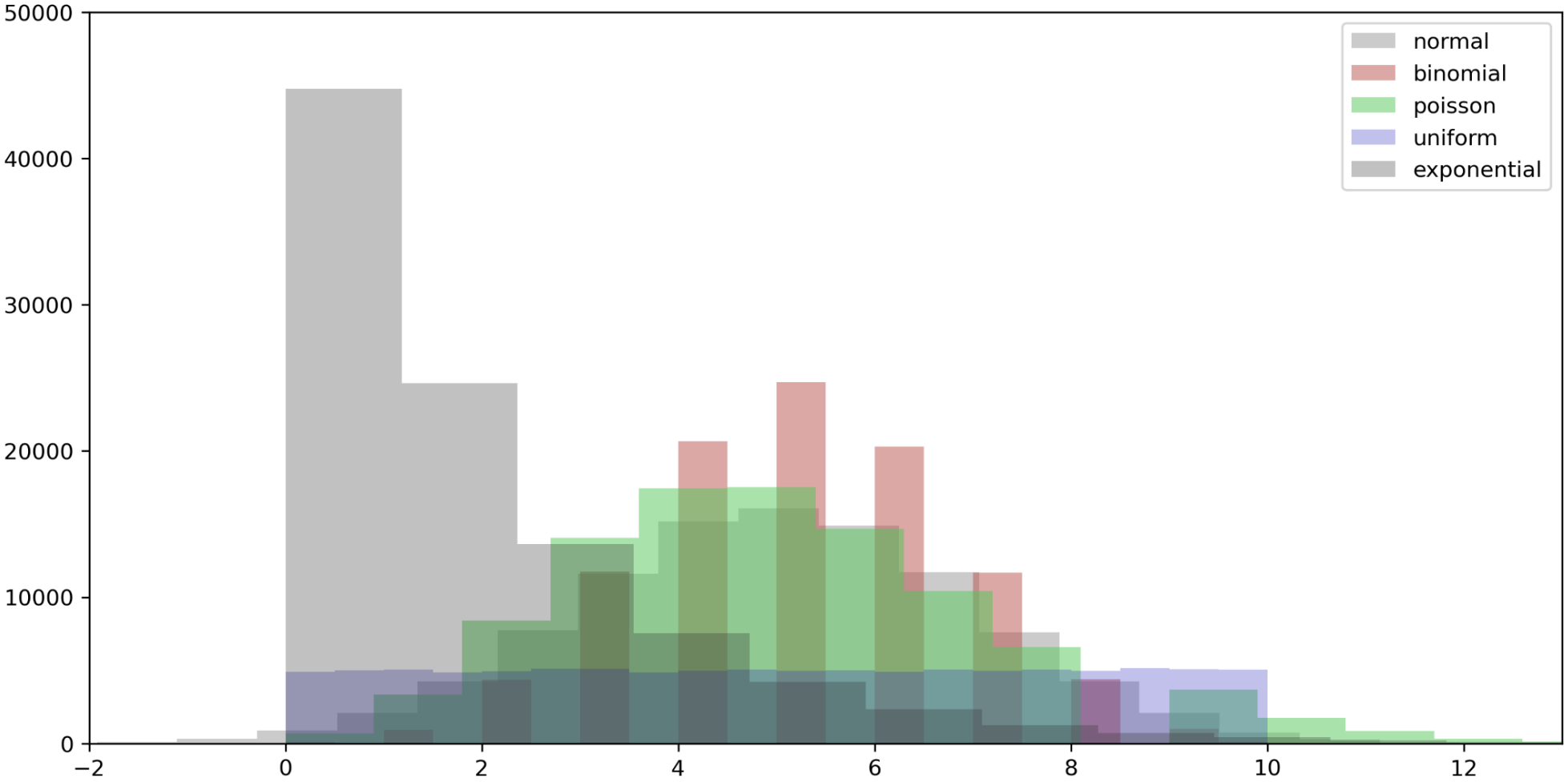
NumPy – Central Limit Theory

- After demonstrating normal distribution, you may try other distributions as you wish.

```
plt.figure(figsize=[12,6], dpi=300)
plt.hist(random.normal(loc=5, scale=2, size=100000), bins=20, color=(0.6, 0.6, 0.6, 0.5),
         label='normal')
plt.hist(random.binomial(n=10, p=0.5, size=100000), bins=20, color=(0.8, 0.3, 0.3, 0.5),
         label='binomial')
plt.hist(random.poisson(lam=5, size=100000), bins=20, color=(0.2, 0.8, 0.3, 0.5),
         label='poisson')
plt.hist(random.uniform(size=100000)*10, bins=20, color=(0.2, 0.2, 0.8, 0.3),
         label='uniform')
plt.hist(random.exponential(scale=2, size=100000), bins=20, color=(0.2, 0.2, 0.2, 0.3),
         label='exponential')
plt.legend()
plt.axis([-2, 13, 0, 50000])
plt.show()
```

NumPy – Central Limit Theory

When **n** is very large, …



The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw

Website: https://toodou.github.io/